An Emergent Approach to Systems of Physical Agents

Position Paper for AAAI Spring Symposium on Software Architectures for Physical Agents

H. Van Dyke Parunak
Industrial Technology Institute
(313) 769-4049 (v), 4064 (f), van@iti.org

1. Background

For the past ten years I have been involved in industrial research on applications of agent architectures to manufacturing, and am presently techical advisor to an industrial consortium that is fielding pilot tests of shop-floor agents. The specific problem addressed by most of our work is the management of material flow and resource utilization on the factory floor. Classically, these problems have been considered under the rubrics of "scheduling" and "shop-floor control," and have been addressed with top-down control that does not offer the robustness and agility required in modern competitive environments. Our approach has been to reify parts and equipment as agents, and seek architectures that permit the overall shop schedule to emerge dynamically from their interaction, rather than being imposed top-down.

Section 2 clarifies the sense in which I use the word "agent." Section 3 briefly summarizes the implementations on which my observations are based. Section 4 outlines five hypotheses that we have derived from these implementation. Section 5 responds directly to the questions posed in the call for participation.

2. What is an "agent"?

"Agent" is an overloaded term among computer scientists. I find it most useful to use the term to describe an active software object (that is, an object with its own thread of control), together with any physical entity (such as a person, a machine, or a part) that the object represents to the rest of the system. This definition clarifies the relation of agents to previous software technologies (Table 1).

	Structured Programming	**Object-Oriented Programming**	**Agent-Oriented Programming**
		Implementation Details	
Structural Unit	Subroutine	Object	Agent
Relation to Previous Level		Subroutine + Persistent Local State	Object + Rules or Goals (independent execution thread)
		Localization Implications	
How does a unit behave? (Code)	Local	Local	Local
What does a unit do when it runs? (State)	External	Local	Local
When does a unit run?	External (called)	External (message)	Local (rules; goals)
		Performance Implications	
Maturity **Fielded systems?** **..Development env'ts?** **..Professionals?**	High Many large Abundant Many experienced	Moderate Some modest Some accepted ones Few experienced	Low: Only a handful Mostly research Mostly researchers
Challenges	Design-time bindings. Systems expensive and hard to maintain.	Issues of system size and integration across vendors and platforms.	Need to move from research to industry.
Promises	Supports extensive legacy systems.	Interoperability; Configurability	Configurability; Adaptability

Table 1: Agents and Other Software Technologies

From the perspective of this definition, several concepts that are often discussed as defining criteria for agents are really incidental characteristics. (Coen 1994 offers a useful survey of some other definitions.)

- Agents do not need to speak any particular language, and in fact agent architectures can be conceived in which agents never communicate directly with one another, but only interact via side-effects in the environment.
- Agents need not emulate human-level intelligence. I am more interested in the complex behavior that emerges from the interactions of many agents (whether simple or complex) than in how sophisticated any single agent can be.
- Agenthood does not depend on an entity's representing ("being an agent for") a human user.
- Agents do not need to model either themselves or their associates explicitly.

While these points distinguish our work from that of many researchers, they do permit us to draw on a wide range of natural systems for inspiration. Ants, cockroaches, and birds do not model one another or support human-level reasoning, but they solve complex problems in the physical world in a modular, distributed, and extremely robust way, and "active objects" seems to be a useful level of characterization for computer imitations.

3. Bases for answers:

The experimental bases for the positions in this paper include experimental systems constructed in our laboratory [summarized in Parunak 1990], ongoing industrial pilots, and a review of applied agent-based systems that includes work by other researchers.[Parunak 1995].

Our initial system, YAMS ("Yet Another Manufacturing System" [Parunak 1987]), decomposes a flexible manufacturing cell into a conventional hierarchical structure, with the Cell made up of several Workstations, each made up of several Devices (e.g., load/unload robot, gauging station, machining resource). We assign agents to each of the nodes in this tree. An agent's task in general is to acquire a task through negotiation with higher-level agents, decompose any task assigned to it into constituent steps at the next lower level, and use the contract net protocol to identify and manage the next lower-level agent to carry out each subtask. The system works, but the hierarchical structure generates an intolerable communications burden.

Our next system, CASCADE, focuses on the material transportation system (a switched conveyor). CASCADE began as another contract net, this time not through a hierarchy but between producers and consumers of pallets of material. Because of the restricted domain, the protocol became extremely stylized, and we realized that the system was homomorphic to a neural network with backpropagation [Parunak et al. 1987]. The system maintains no global map (other than the physical connectivity of the material handling equipment itself), and agents interact only with their nearest neighbors, but the aggregate is able to learn where required parts can be found and to develop its own routings as part locations change. Refinements of this system include development of a layered material handling protocol modeled on the layered architectures that are now commonplace for information networks [Parunak and Judd 1988].

Recently, I have been serving as technical consultant to a number of industrial pilots of agent technology. For proprietary reasons, details of these systems cannot be published, but my experience with them is reflected in my generalizations and hypotheses. I draw also on an extensive review of more than forty industrial systems with agent-like characteristics [Parunak 1995]. This review discusses the applications to which agents are applied in each system, the architecture of individual agents (including their diversity and the level of sophistication of their reasoning), and the architecture of the overall system.

4. Hypotheses for Physical Agents

Our experience leads to the following hypotheses, which I look forward to discussing in more detail with the participants at the workshop. Several of these hypotheses are particularly important for agents in physical domains, and might be overlooked by those whose agents live entirely in networks.

4.1. Agents should reify Things, not Functions.

Classical instincts of software engineers can too easily lead to a functional decomposition of the problem, even when the software modules are instantiated as agents. In manufacturing, such a decomposition requires each function to know the details about many

of the entities in the shop. As a result, the software is tightly coupled to the specific problem, and modification to fit a different problem is costly and time-consuming. If on the contrary we center agents on physical things (such as machines, parts, and people), it appears to be possible to give them fairly generic knowledge about the kinds of functions (e.g., scheduling, material management) that the shop must support, such that any collection of the appropriate things (and thus agents) will emergently support the required functions.

4.2. Agents should be small in "mass," time, and space.

Mass (measured perhaps in lines of code): Each agent should constitute a small part of the entire system, and there should be many of them. Small agents are less costly to implement; easier to understand; and can be made redundant so that the impact of individual failure is limited. In addition, the industrial control marketplace (including vendors such as Rockwell Allen-Bradley, Mitsubishi, Echelon, and Zworld) is moving toward small control computers (8 or 16 bit CPU's with RAM measured in KB rather than MB and on the order of 10 I/O points). Coupled with controller-area networks, these products offer an early entry point for widespread industrial deployment of agents ... if the agents are small enough to fit on them!

Time: Agents should be able to forget (an often-overlooked aspect of learning).

Space: Agents should not be required to maintain or access a global model of the entire system, and their actions should be restricted to their immediate vicinity. I expect this claim to be particularly controversial, and indeed, in purely network applications, one can make a case that the distinction between "near" and "far" is not well defined. But at least in physical systems (and probably in network applications as well), local sensing and action are important for several reasons:

- Remote sensing and action pose risks for agent-based systems analogous to those presented a generation ago by "goto": they yield an incomprehensible tangle of interactions that is difficult to engineer for applications.
- Agent-based systems are susceptible to complex dynamic behavior. Kaufmann's results with NK systems [Kaufmann 1993] suggest that one of the conditions for such collections to self-organize (as opposed to chaotic thrashing) is that interconnectivity be sparse, not dense. Requiring locality is one disciplined way of achieving such sparseness.
- We want long-range interactions to emerge through the interactions of many agents, in order to obtain the robustness and agility characteristic of emergent systems.

4.3. Agents should be redundant.

Systems should accommodate redundant agents that attempt to support the system objectives in different, competing ways. This principle, enabled by light-weight agents, permits the overall system to develop as an ecology and thus evolve to accommodate a changing business environment.

In manufacturing, there are limits to the amount of redundancy that can be tolerated economically. (When a raw casting for a single machined part costs on the order of $100K, few companies can afford turning ten machining stations loose on ten castings in order to generate a single finished product.) However, much of the benefit of redundancy and competition can be achieved if a detailed simulation of the system is run in parallel with the system itself, against the same stream of input data that the real system sees. Modified agents can be tested in the simulated system, and successful competitors can then be swapped into the real system, permitting continuous, incremental improvement.

4.4. Agent engineering should focus on the community, not just the agent.

The previous three hypotheses incline us toward a focus on the aggregate behavior of a community of agents, not just the behavior of an individual agent. In fact, agents are for us just a means to an end, the end being the overall system performance. Natural systems achieve complex robust behavior, not through a single complex intelligence, but through the emergent properties of large populations of simple agents, and we seek to emulate this pattern. For us, questions of intention, mission, and performance are most meaningful at the level of the system as a whole, and need not be reflected explicitly in individual agents.

4.5. Agents may generate, and should monitor, side effects

Physical agents (as opposed to knowbots and other agents that live exclusively in the electronic world) often exist in order to generate physical effects that from a computational point of view are side effects. In

addition, physcal agents must monitor their entire environment, not just electronic messages. In an early implementation of YAMS, a provably "deadlock proof" set of agents deadlocked because the movement of a physical part between agents constituted a "message" that had not been accounted for in the formal analysis of the system's liveness. Often, it is cheaper and more robust to pass information through the physical environment, perhaps even embedded in the product, than to use the kinds of electronic messaging that we computer scientists tend to think of first.

5. Answers to Questions

5.1. Coordination

How should the agent arbitrate/coordinate/cooperate its behaviors and actions? Is there a need for central behavior coordination?

YAMS had hierarchical architecture, requiring two agents that wished to interact to find one another by negotiating up to a common manager and then back down to the intended peer. As in human organizations, this structure led to excessive traffic that bogged down the network, giving unacceptable performance. CASCADE had a flat communication structure in which suppliers and consumers communicated directly with each other. This structure was fixed by the physical connectivity of the system. Because agents communicated only with agents that were "adjacent" to them in the system topology, traffic stayed under control.

At the least, centralized control can lead to unnecessarily complex traffic problems. As discussed under "Hypotheses," it can also lead to building too much domain specificity into each individual agent, so that the system becomes less adaptable to changing circumstances. At the same time, some degree of overall coordination may be needed to insure that systems do not settle into local optima that are globally suboptimal. Our current work emphasizes agents that act autonomously within a local subset of the overall system, supplemented by "watchdog" agents that can see larger regions of the system but whose effects are limited to posting status reports. In "generate and test" parlance, the system state is generated emergently from many locally autonomous agents, but tested by global watchdogs.

5.2. Interfaces

How can human expertise be easily brought into an agent's decisions?

We embody human expertise in an agent by embedding a human in the agent, not by trying to solve the Turing problem. When a human-level decision needs to be made, the agent asks a human to make it.

Will the agent need to translate natural language internally before it can interact with the world?

Our work emphasizes agents as active objects whose computational capabilities may be extremely simple, often far below the level needed to support NLP. While a natural language interface would be nice, our first priority is to map agents onto operational tasks such as shop floor control. Current interface technologies (menus and push-button panels) are adequate for now in the factory environment.

How should an agent capture mission intentions or integrate various levels of autonomy or shared control?

In general, we try to design mission intentions into the overall set of agents rather than into any single agent, so that the dynamic attractor of the community's behavior reflects those intentions. In practice, watchdog agents may be assigned to represent critical functions, but in our experience too much emphasis on "functions" or "intentions" leads to a design that is brittle and not readily transferred to other domains.

Can restricted vocabularies be learned and shared by agents operating in the same environment?

Our agents do not learn vocabularies. They communicate with one another in extremely restricted languages, and (importantly) through the changes they generate in the physical environment.

5.3. Representation

How much internal representation of knowledge and skills is needed? How should the agent organize and represent its internal knowledge and skills? Is more than one representational formalism needed?

Our emphasis on very simple agents reduces the importance of this question, since we do not require agents to model anything. Although YAMS and CASCADE both employed agents all of whom had the same internal structures, this homogeneity was an unnecessary constraint, and systems such as ARCHON/GRATE [Jennings et al. 1992] show that

systems with very different internal representations can be integrated successfully if they are provided with "heads" that all speak the same language.

5.4. Structural

How should the computational capabilities of an agent be divided, structured, and interconnected? What is the best decomposition/granularity of architectural components? What is gained by using a monolithic architecture versus a multi-level, distributed, or massively parallel architecture? Are embodied semantics important and how should they be implemented? How much does each level/component of an agent architecture have to know about the other levels/components?

Hypotheses 1-4 above directly address these issues:

1. Agents should reify things, not functions.
2. Agents should be small in "mass," time, and space.
3. Agents should be redundant.
4. Agent engineering should focus on the community, not just the agent.

5.5. Performance

What types of performance goals and metrics can realistically be used for agents operating in dynamic, uncertain, and even actively hostile environments?

Performance goals in general are associated with the system as a whole, not with a single agent; agents are successful just when their community is successful. Varieties of the bucket-brigade algorithm [Holland 1992], in which successful aggregates distribute their rewards to their members, seem the most promising mechanism for tracking the fitness of individual agents, where a genetic learning scheme makes such tracking important.

How can an architecture make guarantees about its performance with respect to the time-critical aspect of the agent's physical environment?

In general, system performance must be engineered into the attractor of the community's dynamics through thorough simulation during development. Watchdog agents can raise flags when the system deviates from its design goals, and for the foreseeable future human intervention will be the required in such situations (as it is with current systems).

What are the performance criteria for deciding what activities take place in each level/component of the architecture?

Our current architectures assign functions on the basis of the physical entity that an agent represents. Because we eschew functional agents (except for watchdogs), the physical structure of the system constrains what activities are assigned to which agent. The performance that we seek to optimize by this structure is the performance of a human who has to maintain and update the system, and the ability of the system itself to be robust in the face of environmental change, rather than short-time performance objectives such as throughput or WIP.

5.6. Psychology

Why should we build agents that mimic anthropomorphic functionalities? How far can/should we draw metaphoric similarities to human/animal psychology? How much should memory organization depend on human/animal psychology?

Our work does not require anthropomorphized agents. Agents are valued as a highly modular, distributed way to construct a system whose behavior emergently supports changes in the environment, not as consorts to humans. We resort to insects more than to humans for architectural inspiration.

5.7. Simulation

What, if any, role can advanced simulation technology play in developing and verifying modules and/or systems? Can we have standard virtual components/test environments that everybody trusts and can play a role in comparing systems to each other? How far can development of modules profitably proceed before they should be grounded in a working system? How is the architecture affected by its expected environment and its actual embodiment?

Simulation has two uses for agent-based systems: as a development environment, and as a mechanism for improving the behavior of an implemented system.

Simulation is needed because, in general, the interactions of a community of agents cannot be predicted analytically in advance. The aggregate behavior even of very simple agents (for example, boolean networks or cellular automata) is known to be extremely complex, and capable of formally chaotic behavior, and the only known way to determine whether a set of agents will yield the desired overall effect is to simulate them.

Simulation continues to be used during the system's life, to support continuous improvement as operating conditions change. Safety and economic

considerations may preclude experimenting with changes in the agents that operate physical machinery, but we can experiment with a parallel set of agents living in a simulated domain and seeing the same input that the physical system sees. Such experiments, either manual or under a genetic algorithm, show when improvements can be made over the original set of agents. When improvements are validated, they can quickly be swapped into the physical enviornment.

5.8. Learning

How can a given architecture support learning?

CASCADE implemented simple parametric learning through a back-propagation mechanism. Genetic techniques offer promise for more advanced agent learning, but for reasons of safety and economics this learning must often be done in a simulated environment, and then uploaded periodically to the working environment.

How can knowledge and skills be moved between different layers of an agent architecture?

The notion of "layers" of agents presupposes a hierarchical structure that in general we find unnecessary and even burdensome.

References

[Coen 1994] M.H.Coen, "SodaBot: A Software Agent Environment and Construction System." MIT AI Lab AI Technical Report 1493.

[Jennings et al. 1992] Jennings,N.R., Mamdani,E.H., Laresgoiti,I., Perez,J. and Corera,J. (1992), "GRATE: A General Framework for Co-operative Problem Solving." IEE-BCS Journal of Intelligent Systems Engineering 1:2 (Winter), 102-14.

[Holland 1992] J.H.Holland, Adaptation in Natural and Artificial Systems. Second edition, MIT Press.

[Kaufmann 1993] S.A.Kauffman, The Origins of Order: Self Organization and Selection in Evolution. Oxford University Press.

[Parunak 1987] H.V.D.Parunak, "Manufacturing Experience with the Contract Net." In M.N.Huhns, ed., Distributed Artificial Intelligence, Pitman, 285-310.

[Parunak 1990] H.V.D.Parunak, "Distributed AI and Manufacturing Control: Some Issues and Insights." In Y.Demazeau and J.-P.Mueller, eds., Decentralized AI, North-Holland, 81-104.

[Parunak 1995] H.V.D.Parunak, "Applications of Distributed Artificial Intelligence in Industry." in O'Hare and Jennings, Foundations of Distributed Artificial Intelligence, Wiley Inter-Science (forthcoming).

[Parunak and Judd 1988] H.V.D.Parunak and R.Judd, "LLAMA: A layered logical architecture for material administration." International Journal of Computer Integrated Manufacturing 1:4, 222-33.

[Parunak et al. 1987] H.V.D.Parunak; J.Kindrick; and B.W.Irish, "Material Handling: A conservative Domain for Neural Connectivity and Propagation," in Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87), 307-11.